Volume 2 Issue 7 $1.75 DECEMBER 1986

"Serving 99'ers Since 1984"

THE SMART PROGRAMMER

ERROR CHECK FOR XBASIC PROGRAM ENTRY

by Tom Freeman

Editor: The Smart Programmer *has long endeavored to provide the best possible format for programs to be keyed in by our readers. The article that follows provides a new means of ensuring that programs get from us to you without errors. While most of the material in* The Smart Programmer *has never appeared elsewhere, this article was found in the public domain. We deemed it so important to the 99/4A community that we offer it to our readers. As is our policy, we obtained permission from the author to reprint the article from the LA 99'ers newsletter (*LA Topics*). This article represents a milestone for the 99/4A community. Our future XBASIC offerings will follow the guidelines of this article.* *RM, Editor*

Have you ever typed in a TI-99/4A version of a BASIC program from a magazine and noticed that the other versions have little numbers at the end of the lines that you don't have? They were for error checking on your typing, to ensure no mistakes. Have you ever laboriously typed in a long program and run it, only to find that it crashes, or doesn't work as it is supposed to, all because of a simple typing error that you can't find? So why doesn't TI have one? NOW YOU DO!!

This may be the most useful program that I have published for general use, because almost everyone does BASIC programs at one time or another. It involves only one extra step for the programmer, and one for the user who is typing the published program in. It is really a rather simple method, and depends on the manner in which TI stores BASIC programs. Please note, however, that it requires a memory expansion and disk drive, and works only in Extended BASIC (although BASIC programs can be entered in XBASIC, SAVEd, and then RUN in BASIC).

You may remember the format in which "MERGE" type programs are stored on disk. If you don't, see our article (*LA Topics*) a couple of months back on the various formats in which programs are stored. The MERGE format is actually a duplicate of the way in which the actual program is stored in memory, or on disk, the difference being that it is a display type file, with each record starting with two bytes for the line number, and then the actual program line. In memory, however, the program lines are stored contiguously, and in seemingly random order (actually the order depends on the order in which they were entered). A separate line number table is stored below the program area and keeps track of the line numbers and pointers to where each line begins. Now each line consists of one byte "tokens" for all reserved words (see the list I published last month in *LA Topics*) with all strings, including the names of subprograms such as LOAD, SCREEN, etc., being spelled directly.

When you enter any line in XBASIC (either a command or a program line with the line number coming first) it is first moved to the so-called "Edit Buffer" at address >8CØ in VDP. The BASIC bias is preserved. The purpose of this is that if you press

FCTN 8 (REDO), then the whole line or lines can be retrieved. Next, everything is "crunched" by replacing each reserved word with its token, subtracting the BASIC bias from strings, computing their length, etc. and placing the result in the "crunch buffer" at >82Ø in VDP. Once it is there, it can be transferred to the appropriate place in memory expansion. This is the area that is used when my program computes the "checksum" by merely adding the value of each byte! The number is never allowed to go over hex >FF -- the high byte is ignored (thus, in decimal, no number over 255). The assumption is that it is extremely unlikely, probability approaching zero, that a small number of mistakes will result in a number that differs by exactly 256 or a multiple thereof. The one exception is that if you transpose two characters, there's nothing I can do about that!

Now what does the programmer do? First, his program must be completely debugged, as no changes can be made after the checksums are computed, or they will of course differ. Next he SAVEs his program in MERGE format. Now, the following program must be run on the result:

```
> 100 !CREATE CHECKSUMS FOR XB
  ASIC PROGRAMS, BY TOM FREEMA
  N, LA 99'ERS !250
> 110 !SHOULD BE USED TOGETHER
   WITH "CHECK" ASSEMBLY FILE
  THAT WILL PRINT CHECKSUMS ON
   SCREEN !099
> 120 DISPLAY AT(2,1)ERASE ALL
  :"CREATE CHECKSUMS FOR XBASI
  C ERROR CHECKING": :"    by
  Tom Freeman" !085
> 130 DISPLAY AT(10,1):"INPUT
  MERGE FILE?":"  DSK1." !007
> 140 DISPLAY AT(13,1):"OUTPUT
   MERGE FILE?":"  DSK1." !108
> 150 ACCEPT AT(11,3)SIZE(-15)
  BEEP:I$ :: OPEN #1:I$,VARIAB
  LE 163,INPUT !192
> 160 ACCEPT AT(14,3)SIZE(-15)
  BEEP:O$ :: OPEN #2:O$,VARIAB
  LE 163,OUTPUT !053
> 170 DISPLAY AT(20,1):"ANALYZ
  ING LINE":"CHECKSUM IS " !01
  4
> 180 LINPUT #1:A$ :: IF LEN(A
  $)=2 THEN CLOSE #1 :: PRINT
  #2:CHR$(255)&CHR$(255):: CLO
  SE #2 :: STOP !115
> 190 Z=ASC(A$)*256+ASC(SEG$(A
  $,2,1)):: DISPLAY AT(20,15)B
  EEP:Z !141
> 200 B$=SEG$(A$,3,163):: L=LE
  N(B$):: IF L>157 THEN 230 !1
  62
> 210 N=Ø :: FOR X=1 TO L :: Y
  =ASC(SEG$(B$,X,1)):: N=N+Y :
  : NEXT X :: N=N AND 255 :: N
  $=STR$(N):: N$=RPT$("Ø",3-LE
  N(N$))&N$ !088
> 220 DISPLAY AT(21,13)BEEP:N$
   :: PRINT #2:SEG$(A$,1,L+1)&
  CHR$(131)&N$&CHR$(Ø):: GOTO
  180 !252
> 230 DISPLAY AT(22,1)BEEP:"WA
  RNING!":" LINE";Z;"IS TOO LO
  NG!":"PRESS ANY KEY TO CONTI
  NUE" !123
> 240 CALL KEY(Ø,K,S):: IF S=Ø
   THEN 240 ELSE PRINT #2:A$ :
  : GOTO 180 !232
```

Notice the "!" and 3 numbers at the end of each line? The program was RUN on itself! Here is what happens. Each record of the MERGE file is read in, the first two bytes ignored (we don't need the line number) and the rest are added up. Next, the identical record is printed to the output file, with the addition of the token for "!" (REMark) and the 3 characters of the checksum. This will work even if the program line already contained a REMark (as in lines 1ØØ-11Ø). THE USER MUST BE WARNED NOT TO TYPE THESE 4 CHARACTERS, since they were not computed into the checksum. At the end (it may take a little while with a long program, but only needs to be RUN once), the programmer types NEW and MERGEs in the output file, then SAVEs it in normal mode, or lists it to printer, or whatever. This is the form to be published.

Now what the user must do once is type in the source code attached to the end of this article and assemble it (a CALL LOAD version is also supplied for those who don't have the Editor/Assembler). If the object code created was called "CHECK" then he must type the following upon entry into XBASIC: CALL INIT :: CALL LOAD("DSKx.CHECK") :: CALL LINK("CURSOR"). This one line with a line number can be SAVEd on disk and then RUN each time it is needed, rather than type the whole line. What the assembly routine at CURSOR does is some housekeeping such as moving the numbers Ø-9 to character sets 13-14, changing the colors there, redefining the cursor, putting up the title screen, etc. and then turning on the user-defined interrupt. Now at every VDP interrupt (each 1/6Ø second), the routine at CHECK begins. The interrupt

can be turned off with CALL LINK("OFF") and back on with CALL LINK("ON") at any time and the shape of the cursor will tell you which mode you're in. Now, EVERY TIME you enter a new program line (and for some reason after FCTN 8 REDO even if no changes are made) the checksum will appear at the bottom of the screen and one extra line scrolled up. HERE IS THE KEY -- IT SHOULD CORRESPOND TO THE ONE PUBLISHED THAT YOU ARE ATTEMPTING TO C O P Y I N . H e n c e , n o e r r o r s ! ! !

I think the source code is sufficiently commented to explain what is going on. I must add that I spent many hours with MG Explorer, by Doug Warren, finding out WHAT is going on when you enter a line in XBASIC. The address range in GROM of >6AAØ to >6AD8 should be broad enough to cover the various versions of XBASIC out there, since they differ by a few bytes here and there (the actual range needed in my module was >6AAE to >6ACA. This area contains the loop where the first key press on entry of a new line is located. As soon as the first key is pressed, then the GROM code moves on. I needed this area so as to reset the flag that indicates the checksum has been printed, in order to avoid having it printed again and again! Notice the fairly cumbersome method of peeking at the GROM address, which must then be reset, since just looking at it destroys it! I discovered that the line number entered is SAVEd at BOTH >83Ø4 and >834A and only when it is at both is the crunch buffer finished being filled with the crunched line. If you are entering a direct command, >83Ø4 is not used until much later, which is why I clear it at the beginning of each entry, so the routine won't get confused.

Finally, if all the criteria are met, >83Ø4 = >834A and KEY (>8375) contains the valid entry key (enter = >ØD, up arrow = >ØB or down arrow = >ØA), then the meat of the program goes to work, computes the checksum and puts it on the screen after an extra scroll (XBASIC does its own scroll after I'm finished). Please note that I use BLWP @XMLLNK with DATA SCROLL instead of adding the whole routine. This saves a lot of typing. However, for those of you who are interested, I am also providing the entire routine done by DISkASSEMBLER™, so that you can place it in an E/A assembly file if you wish, as this one exists in Bank 1 of XBASIC's ROM at >6ØØØ->7FFF, and hence can't be used by E/A.

I'm hoping that everyone finds this program useful and that it is widely used. I'm only sorry I didn't write it three years ago! Finally, I would like to thank Doug Warren for writing Explorer, without which I could not have done this, since I needed to find out where XBASIC does what! (I also must blame Doug for my bleary eyes!) And, I especially would like to thank Craig Miller for his invaluable help and advice while I was writing the program. As Craig slowly leaves the TI community, we will all feel the loss.

```
> 1 !CALL LOAD VERSION OF OBJE
  CT CODE FOR CHECKSUM PROGRAM
  ,BY TOM FREEMAN,LA 99ERS !20
  Ø
> 100 CALL INIT :: CALL LOAD(9
  460,0,0,0,0,0,0,106,160,106,
  216,0,10,11,13,0,0)!180
> 110 CALL LOAD(9484,0,126,66,
  66,66,66,126,0,31,31,32,32,8
  8,66,65,83,73,67,32,69)!144
> 120 CALL LOAD(9504,82,82,79,
  82,32,67,72,69,67,75,69,82,3
  2,32,32,32,32,32,85,83,73,78
  )!107
> 130 CALL LOAD(9526,71,32,67,
  72,69,67,75,83,85,77,83,32,3
  2,32,32,32,66,89,32,84,79,77
  )!119
> 140 CALL LOAD(9548,32,70,82,
  69,69,77,65,78,44,32,76,65,3
  2,57,57,69,82,83,2,132,0,10)
  !052
> 150 CALL LOAD(9570,17,2,2,36
  ,0,7,2,36,0,48,192,68,2,33,0
  ,176,6,193,4,32,32,32)!199
> 160 CALL LOAD(9592,4,91,2,0,
  3,240,2,1,37,4,2,2,0,8,4,32,
  32,44,2,0,4,128)!121
> 170 CALL LOAD(9614,2,1,39,22
  ,2,2,0,80,4,32,32,44,2,0,7,0
  ,4,32,32,36,4,32)!166
> 180 CALL LOAD(9636,32,24,0,3
  8,2,2,37,22,2,3,96,96,2,4,0,
  36,192,66,172,131,6,4)!204
> 190 CALL LOAD(9658,22,253,2,
  0,2,228,2,2,0,24,4,32,32,36,
  4,32,32,24,0,38,2,0)!067
> 200 CALL LOAD(9680,2,228,2,1
  ,37,46,2,2,0,24,4,32,32,36,4
  ,32,32,24,0,38,2,0)!020
> 210 CALL LOAD(9702,2,228,2,1
  ,37,70,2,2,0,24,4,32,32,36,2
  ,0,3,240,2,1,37,12)!006
> 220 CALL LOAD(9724,2,2,0,8,4
  ,32,32,36,2,0,38,36,200,0,13
  1,196,4,91,2,0,3,240)!119
```

```
> 230 CALL LOAD(9746,2,1,37,4,
  2,2,0,8,4,32,32,36,4,224,131
  ,196,4,91,216,32,152,2)!239
> 240 CALL LOAD(9768,36,248,6,
  224,36,248,216,32,152,2,36,2
  48,6,224,36,248,6,32,36,248,
  136,32)!133
> 250 CALL LOAD(9790,36,248,36
  ,250,26,8,136,32,36,248,36,2
  52,27,4,4,224,36,244,4,224,1
  31,4)!013
> 260 CALL LOAD(9812,216,32,36
  ,248,156,2,6,224,36,248,216,
  32,36,248,156,2,2,0,8,28,2,1
  )!054
> 270 CALL LOAD(9834,37,20,2,2
  ,0,2,4,32,32,36,2,0,8,15,2,1
  ,244,0,2,2,0,13)!105
> 280 CALL LOAD(9856,4,32,32,3
  2,5,128,6,2,22 ,1,2,0,7,4,4
  ,32,32,48,7,96,36,244)!204
> 290 CALL LOAD(9878,22,62,2,1
  ,0,3,152,33,36,254,131,117,1
  9,3,6,1,22,250,4,91,200,32)!
  180
> 300 CALL LOAD(9900,131,4,131
  ,4,19,49,136,32,131,4,131,74
  ,22,45,7,32,36,244,208,160,1
  31,66)!038
> 310 CALL LOAD(9922,9,130,2,0
  ,8,32,2,1,39,22,4,32,32,44,4
  ,224,37,2,184,49,37,3)!195
> 320 CALL LOAD(9944,6,2,22,25
  2,200,11,36,246,4,32,32,24,0
  ,38,2,0,2,226,193,96,37,2)!1
  38
> 330 CALL LOAD(9966,2,2,0,10,
  2,3,0,100,2,6,0,2,4,196,61,3
  ,6,160,37,94,5,128)!027
> 340 CALL LOAD(9988,192,194,6
  ,6,22,248,193,5,6,160,37,94,
  194,224,36,246,4,91)!104
> 350 CALL LOAD(16376,79,78,32
  ,32,32,32,37,244)!042
> 360 CALL LOAD(16368,79,70,70
  ,32,32,32,38,14)!240
> 370 CALL LOAD(16360,67,72,69
  ,67,75,32,38,36)!002
> 380 CALL LOAD(16352,67,85,82
  ,83,79,82,37,122)!053
> 390 CALL LOAD(8194,39,22,63,
  224):: CALL LINK("CURSOR")!1
  43
```

```
* SCROLL ROUTINE -- FOR USE IN
*                   OTHER PROGRAMS
*
* WORKSPACE MUST BE >83E0
*
SCROLL LI   R12,>02E0
       LI   R10,>0020
       CLR  R9
       MOV  R11,R6
       BL   @AA
       LI   R5,>8C00
       LI   R4,>02E0
       LI   R1,>7F80
       LI   R2,>001C
       BL   @AF
       MOVB R1,*R5
       SWPB R1
AB     MOVB R1,*R5
       DEC  R2
       JNE  AB
       SWPB R1
       MOVB R1,*R5
       MOVB R1,*R5
       B    *R6
AA     CLR  R8
       MOVB @>83F5,*R15
       STWP R7
       MOVB R10,*R15
AD     MOVB @>8800,*R7+
       INC  R10
       INC  R8
       DEC  R12
       JEQ  AC
       CI   R8,>000C
       JLT  AD
AC     MOVB @>83F3,*R15
       ORI  R9,>4000
       MOVB R9,*R15
       STWP R7
AE     MOVB *R7+,@>8C00
       INC  R9
       DEC  R8
       JNE  AE
       MOV  R12,R12
       JNE  AA
       B    *R11
AF     MOVB @>83E9,*R15
       ORI  R4,>4000
       MOVB R4,*R15
       NOP
       MOVB R1,@>8C00
       B    *R11
```

```
* SOURCE CODE TO WRITE CHECKSUM FOR ENTERED XB LINE ON SCREEN
* BY TOM FREEMAN, LA 99ERS
* THIS IS PUBLIC DOMAIN, PLEASE DISTRIBUTE IT WIDELY!
       DEF  ON,OFF,CHECK,CURSOR
VMBR   EQU  >202C
VMBW   EQU  >2024
```

GENEVE t.m.

MODEL 9640 FAMILY COMPUTER

This unit is without a doubt the most sophisticated machine ever offered in the family and small business area to date. With over a year of design and development, including input from more than one hundred users, this machine has surpassed even our own expectations. Take a moment to review some of the many features that place this computer in a class of its own.

* **99/4(A) COMPATIBLE RUNS OVER 100 EXISTING TI CARTRIDGE PROGRAMS**
* **IBM TYPE KEYBOARD** Included
* **V9938 ADVANCED VIDEO DISPLAY PROCESSOR**

  Is software compatible with TMS9918A (used in 99/4A).

  Uses 46 registers for high speed "HARDWARE" graphics commands.

  Commands include:

DRAW	SEARCH	POINT (status)	BLINK
FILL	MOVE	Animation	And more

  Uses color Pallet of 512 colors on the screen at a single time. 7 modes of graphics operation; some modes allow 256 colors. True BMG (Bit-Mapped-Graphics) operation. Both composite (like the 99/4A) and analog RGB outputs (like the Atari ST and Commodore Amiga). Supports up to 256 colors per screen in the 256 by 424 mode or 16 colors in the 512 by 424 mode. Comes with 128K bytes of video RAM (8 times the amount of the 99/4A)

* **Software Support Supplied with the 9640**

  MYARC DOS (similar to MS-DOS 2.1)

  MYARC ADVANCED BASIC

  —Compatible with TI Extended BASIC and MYARC Extended BASIC II

  —Supports all modes of the Video Processor (including 80 column)

  —Supports Windows

  —Supports easy to program Mouse Commands

  —Combined Text and Bit-Mapped-Graphics Modes

  —Drawing Commands such as Circle, Rectangle, etc. are built-in

  Program patches to make TI-Writer 1) more powerful 2) display 80 columns

  Program to SAVE your 99/4A cartridges to disk

* **99/4(A) COMPATIBLE RUNS OVER 95% OF ALL ASSEMBLY LANGUAGE PROGRAMS & UTILITIES**
* **REAL TIME CLOCK CHIP**
* **MULTIPLAN ALSO 80 COLUMNS**
* **SOUND CHIP COMPATIBLE WITH 99/4A** (3 simulataneous tones, 1 noise)
* **FASTER**—At least 3-4 times

  TMS 995 U-Processor

  Runs same instruction set as 9900 used in 99/4A **plus** 4 new ones. Pipelined processor (i.e. u-processor performs several functions **simultaneously).**

* **Awesome amount of RAM**

  512K of CPU RAM (user configurable between CPU-RAM, RAM-DISK or PRINT-SPOOLER). Expandable to 1 megabyte with MYARC 512K Card. In 99/4A mode 64K of the 512K becomes GROM and 16K CARTRIDGE ROM

* **TI 9995 Processor Chip—12 MHz**

  256 Bytes ULTRA High Speed on Chip RAM. Prefetch on Instructions. Post-store on Instructions.

* **Built-In Mouse Interface**

  Installed hardware allows for the MS mouse to be connected directly to the 9640 board. Basic language support for the mouse built in. Uses the industry-standard MacIntosh mouse commands.

* **Hardware and Software Support for the most commonly-used peripherals**

  Floppy Disk Controllers include MYARC, Texas Instruments, and Corcomp. RS232 cards include MYARC, Texas Instruments, and Corcomp. Ram-Disks include Horizon.

MasterCard

CALL FOR THE BEST PRICES!!

VISA

Disk Only Software
P.O. Box 244
Lorton, Virginia 22079

or call

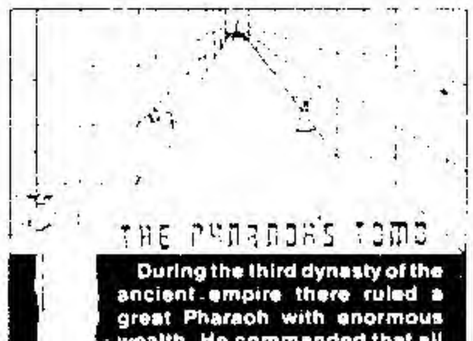
1-800-446-4462. At the tone, enter 897335 for recorded order message. Touchtone phone is required

Alternate is **(301) 369-1339.** No Touchtone is required.

Delphi: TELEDATA—CompuServe: 74405,1207—MCI: TDG—TELEX: 6501106897 MCI

BATTLE OVER TITAN

Your short range scanner has located Torg craft that are out to destroy your outpost on Titan. Time is crucial! Can you maneuver your Drone through the diminishing weak areas in the Torgs scanner and weapon aiming systems. How long can you stave off the Torgs attack? High speed, fast action, special sound effects, high resolution full color graphics, 10 levels of difficulty, random game boards — no two games play alike. Will you be able to repair additional fighter drones in time to save your outpost?



During the third dynasty of the ancient empire there ruled a great Pharaoh with enormous wealth. He commanded that all his wealth shall be placed in a special tomb that was built over a very deep pit.

The myths about the tomb tell us of trap doors and walls that move. We've also heard that the kings ghost is still there protecting his treasure to this day! Automatic multi-level with free plays. High resolution full color graphics. Special sound effects. The Tombs are different each time you play! How long can you last in the Pharaohs Tomb?

Casino Black Jack has been designed to teach you when to stand, hit, split pairs and double down. It's like having a professional Black Jack player helping you increase your odds of winning by telling you what to do depending upon the dealers up card. Along with its high resolution color graphics you have the following options: stand, hit, double down, split a pair and insurance.

Great practice for card counters as you can choose to be dealt from a single deck, 2 decks or 4 decks. Three modes of play include, teach mode, test mode and the play only mode. Hours of educational fun!

"Fifteen men on the dead mans chest Yo-Ho-Ho, and a bottle of Rum."

Your five man diving team has found the lost treasure of Blackbeard the pirate! Unfortunately it is at the bottom of a shark infested sea and it is protected by two giant Octopuses!!

**************** Warning ****************

The sharks haven't been fed since the last time someone played this game and they love to eat divers!!!

Automatic multi-level game with free plays. Beautiful full color graphics and special sound effects. Written in Extended Basic.

THE CRAZY FUN HOUSE

Enter the Crazy Fun House at your own risk where Smiley, Grouchy and their 6 friends are waiting for you to make a wrong turn. This crazy game has visible and invisible passageways where you can chase or be chased, dodge, shoot and rack up points for extra men. It pushes your TI 99/4, 99/4A in extended basic to its ultimate limits with 9 screens and 32 levels of fast action, colorful graphics and superb sound effects.

Optional test mode allows you to start at any level! Joysticks and extended basic are requ

Can you learn the secret patterns of each screen?

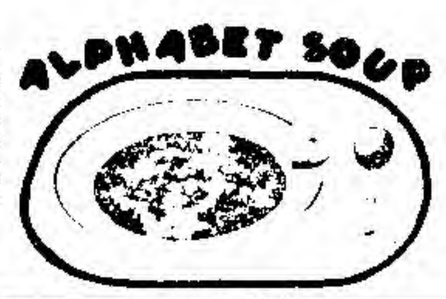
SPECIAL

MG GAMES

All of the above 6 Games on one Menu—Driven disk

ONLY $20

(X—Basic required, Includes shipping)



At last a cracker that won't get soggy. Souper Cracker to the rescue. CRUNCH CRUNCH GOBBLE GOBBLE — Out of the soup the letters fly. This fast action game was designed for the younger computer user, with a prompt mode that displays the next letter to be eaten. Full of rewards and colorful graphics. Eating soup has never been this much fun. Optional speech. Joysticks and extended basic required.

OTHER MG ITEMS AVAILABLE

Smart Programming Guide For Sprites	SPGS	6.95
The Orphan Chronicles	B007	9.95
Advanced Diagnostics	UT01	19.95
DISkASSEMBLER	UT03	19.95
Night Mission — Disk	G10D	19.95
Night Mission — Cass	G10C	19.95
Spare Gram Kracker Manual	GK04	2.50 *
Gram Kracker Battery	GK05	2.50 *
Gram Kracker Utility I Disk	GK03	10.00 *
Prom Set For Corcomp Disk Controller Card	PR01	34.95 *
Video Information Directory Program — IBM	VID1	59.95
Video Information Directory Program — TI PRO.	VID2	59.95

* Price Includes Shipping & Handling — All Other Items Please Add 2.00 per Order

MG 1475 W. CYPRESS AVE., SAN DIMAS CA 91773 — (714) 599—1431

Order with a Check or Money Order or contact your favorite dealer

```
VSBR    EQU  >2028
VSBW    EQU  >2020
VWTR    EQU  >2030
XMLLNK  EQU  >2018
SCROLL  EQU  >0026          ADDRESS OF ROUTINE IN ROM INDEXED ON >6010
NSAVE   EQU  >8304                    EQU >7ADA IN MY XB MODULE
LSAVE   EQU  >8342          ADDRESS WHERE LENGTH OF CRUNCHED LINE IS SAVED
FAC     EQU  >834A
GRMRA   EQU  >9802          GROM READ  ADDRESS PORT
GRMWA   EQU  >9C02          GROM WRITE ADDRESS PORT
DONE    DATA 0
SAV11   DATA 0
SAVEGA  DATA 0
LOWAD   DATA >6AA0          /ADDRESS RANGE IN GROM WHERE FIRST KEY PRESS
HIAD    DATA >6AD8          \ON COMMAND LINE IS REQUESTED
ENTER   DATA >000A,>0B0D    ENTER KEY, UP AND DOWN ARROW
COUNT   DATA 0
CUR1    BSS  8
CUR2    DATA >007E,>4242,>4242,>7E00 HOLLOW CURSOR DATA
INVVID  DATA >1F1F          INVERSE VIDEO COLORS
TITLE1  TEXT '  XBASIC ERROR CHECKER  '
TITLE2  TEXT '    USING CHECKSUMS     '
TITLE3  TEXT 'BY TOM FREEMAN, LA 99ERS'
GETDEC  CI   R4,10          /IF NUMBER IS 10+ THEN NEED TO GET TO >41 ("A"
        JLT  GD             \NOT >3A
        AI   R4,7
GD      AI   R4,>30         MAKE IT AN ASCII CHARACTER
        MOV  R4,R1
        AI   R1,>B0         THIS IS BASIC BIAS OF >60 PLUS >50 TO GET TO
        SWPB R1             TO MSG    ALTERNATE CHARACTER SET AT ASCII 128
        BLWP @VSBW          WRITE ON SCREEN
        RT
CURSOR  LI   R0,>03F0
        LI   R1,CUR1
        LI   R2,8
        BLWP @VMBR          SAVE ORIGINAL CURSOR PATTERN AT CUR1
        LI   R0,>480        /THE 80 BYTES FROM >480 TO >4CF ARE ASCII 48-
        LI   R1,LBUF        |57 ("0" TO "9").  TEMPORARILY STORED AT
        LI   R2,80          \LBUF
        BLWP @VMBR
        LI   R0,>700
        BLWP @VMBW          NOW PUT THEM AT >700 AS ALTERNATE CHAR. SET
        BLWP @XMLLNK
        DATA SCROLL         SCROLL UP 1 LINE
        LI   R2,TITLE1
        LI   R3,>6060       ADD BASIC BIAS TO TITLE CHARACTERS
        LI   R4,36
        MOV  R2,R1
CR1     A    R3,*R2+
        DEC  R4
        JNE  CR1
        LI   R0,>2E4
        LI   R2,24
        BLWP @VMBW          WRITE 1ST LINE
        BLWP @XMLLNK
        DATA SCROLL         SCROLL AGAIN
        LI   R0,>2E4
        LI   R1,TITLE2
        LI   R2,24
        BLWP @VMBW          WRITE 2ND LINE
```

```
        BLWP @XMLLNK
        DATA SCROLL         SCROLL AGAIN
        LI   R0,>2E4
        LI   R1,TITLE3
        LI   R2,24
        BLWP @VMBW          WRITE 3RD LINE
* CALL LINK("CURSOR") DOES THE SETUP AND CONTINUES ON TO "ON"
* CALL LINK("ON") STARTS HERE AND DOESN'T NEED THE SETUP
ON      LI   R0,>03F0
        LI   R1,CUR2
        LI   R2,8
        BLWP @VMBW          LOAD THE HOLLOW CURSOR INTO VDP
        LI   R0,CHECK       LOAD THE INTERRUPT ADDRESS INTO THE ISR
        MOV  R0,@>83C4      \(INTERRUPT SERVICE ROUTINE) HOOK AT >83C4
        RT
OFF     LI   R0,>03F0
        LI   R1,CUR1
        LI   R2,8
        BLWP @VMBW          RELOAD THE ORIGINAL CURSOR
        CLR  @>83C4         CLEAR THE ISR HOOK(TURN OFF INTERRUPT)
        RT
CHECK   MOVB @GRMRA,@SAVEGA "PEEK" AT THE CURRENT GROM ADDRESS AND SAVE
        SWPB @SAVEGA        IT AT SAVEGA, MSB 1ST. GROM ADDRESS IS NOW
        MOVB @GRMRA,@SAVEGA INDETERMINATE
        SWPB @SAVEGA
        DEC  @SAVEGA        ADJUST FOR AUTO INCREMENT
        C    @SAVEGA,@LOWAD TEST FOR THE LOW END OF RANGE WHERE START OF
        JL   CHECK1         COMMAND LINE IS, JUMP OUT IF TOO LOW
        C    @SAVEGA,@HIAD  HIGH END OF RANGE
        JH   CHECK1         JUMP OUT IF TOO HIGH
        CLR  @DONE          RESET FLAG FROM PREVIOUS CHECKSUM ROUTINE
        CLR  @NSAVE         THIS CORRECTS FOR A MYSTERIOUS ERROR I FOUND!
CHECK1  MOVB @SAVEGA,@GRMWA RESET GROM ADDRESS TRHOUGH GRMWA PORT
        SWPB @SAVEGA
        MOVB @SAVEGA,@GRMWA
*NEXT 4 LINES SET THE "INVERSE VIDEO" FOR CHECKSUMS-CAN BE DELETED
        LI   R0,>81C        RESET COLORS FOR CHARACTER SETS 13-14 AT EVERY
        LI   R1,INVVID      INTERRUPT (XB ALWAYS RESETS TO DEFAULT). DELETE
        LI   R2,2           THESE 4 LINES IF YOU DON'T LIKE THE INVERSE
        BLWP @VMBW          VIDEO EFFECT
*NEXT 10 LINES CHANGE SCREEN & CHAR COLORS WHILE IN CHECKSUM MODE
*AND CAN BE DELETED IF YOU DON'T LIKE THE EFFECT
        LI   R0,>80F        START OF COLOR TABLE FOR CHAR SET 0
        LI   R1,>F400       WHITE ON BLUE
        LI   R2,13          13 COLOR SETS
COL     BLWP @VSBW          WRITE A BYTE TO COLOR TABLE
        INC  R0             NEXT COLOR SET
        DEC  R2
        JNE  COL
        LI   R0,>0704       SCREEN COLOR 4(DARK BLUE)
        BLWP @VWTR
*END OF OPTIONAL LINES
        ABS  @DONE          /IF THE ROUTINE WAS ALREADY DONE
        JNE  RETURN         \GET OUTTA HERE!
        LI   R1,3            CHECK FOR THE 3 VALID ENTRY KEYS AND LEAVE IF
CHECK2  CB   @ENTER(R1),@>8375   THERE AREN'T ANY. NOTE USE OF INDEXING
        JEQ  C1             IF VALID KEY THEN GO ON
        DEC  R1             GO FOR MORE
        JNE  CHECK2
        RT
```

```
C1      MOV   @NSAVE,@NSAVE /WHEN >8304 CONTAINS A NON ZERO KEY AND IS =
        JEQ   RETURN        \WHAT IS IN >834A THEN WE'RE READY TO GO!
        C     @NSAVE,@FAC
        JNE   RETURN
        SETO  @DONE         INDICATE THE CHECKSUM IS ABOUT TO BE WRITTEN
        MOVB  @LSAVE,R2     GET THE LENGTH BYTE OF CRUNCHED LINE
        SRL   R2,8          MOVE TO LSB
        LI    R0,>0820      CRUNCH BUFFER
        LI    R1,LBUF       WHERE WE WILL STORE IT
        BLWP  @VMBR         MOVE IT
        CLR   @COUNT        COUNT WILL CONTAIN CHECKSUM, IN BINARY
C2      AB    *R1+,@COUNT+1 /ADD EACH BYTE OF CRUNCHED LINE TO IT, 1 BY 1
        DEC   R2             !BECAUSE WE ARE ADDING BYTES, WHEN WE GO OVER
        JNE   C2             \FF, THE CLOCK GOES BACK TO ZERO
DO      MOV   R11,@SAV11    SAVE THE RETURN ADDRESS
        BLWP  @XMLLNK
        DATA  SCROLL        SCROLL UP THE SCREEN
        LI    R0,>2E2       3RD COLUMN, BOTTOM ROW OF SCREEN
        MOV   @COUNT,R5     MOVE THE VALUE AT COUNT (WORD VALUE BUT LESS
        LI    R2,10         THAN 256, TO R5
        LI    R3,100        R2 AND R3 CONTAIN THE DIVISORS
        LI    R6,2          2 LOOPS FOR 100'S AND 10'S PLACE
D1      CLR   R4            ASL DIVISION IS DONE THIS WAY.VALUE OF 1ST R
        DIV   R3,R4         IS DIVIDED "INTO" 2ND 4(E.G. R3 INTO R4). THE
*                           2ND REG IS ACTUALLY 2 CONTIGUOUS REGISTERS.
*                           THE QUOTIENT IS PLACED IN THE FIRST AND THE
*                           REMAINDER IN THE 2ND.ORIGINALLY THE FIRST MUST
*                           BE 0, OR THERE WILL BE AN "OVERFLOW"
*                           SO R4 NOW CONTAINS THE INTEGER QUOTIENT
        BL    @GETDEC       CONVERT IT TO ASCII AND PUT ON SCREEN
        INC   R0            NEXT SCREEN POSITION
        MOV   R2,R3         NEXT DIVISOR
        DEC   R6            ANY MORE TO DO?
        JNE   D1
        MOV   R5,R4         1'S PLACE IS THE REMAINDER FROM 2ND DIVISION
        BL    @GETDEC       PUT THIS ONE ON SCREEN TOO
        MOV   @SAV11,R11    RESTORE RETURN ADDRESS
RETURN  RT                  AND RETURN
*                           THIS IS END OF PROGRAM AND IS A CONVENIENT PLACE
*                           TO PUT THE BUFFER, WHICH HAS NO DATA TO START
LBUF    END
```

OPTIONAL XB A/L ARGUMENTS

by Richard M. Mitchell

You may have noticed that there are several TI XB statements that allow optional arguments. For instance, CALL HCHAR can include the number of repetitions of a character or the argument can be omitted. User-written Assembly code can also utilize optional arguments.

Once an XB program has LINKed to Assembly code, the address >8312 contains the number of arguments passed, as explained on page 278 of the E/A manual. Thus, a simple compare/jump structure can be employed to make arguments optional.

The type of argument passed can also allow options, such as directing branching, but I have been unable to ascertain from the E/A manual where the argument identifiers reside in the XB environment, as it provides only the locations for BASIC. Well, the location is >83ØØ through >83ØF (thanks to Scott Darling for providing this info). By including a compare/jump structure on the identifiers, some nifty tricks can be employed. For instance, if a base conversion program is being written, a number could

indicate a branch to a decimal to hex routine, with a string indicating a branch to a hex to decimal routine! As detailed on page 278 of the E/A manual, the argument identifiers are numbered Ø through 5.

Here is a simple program that illustrates some of the techniques described in this article (see what you can come up with!):

```
        DEF  EXAMPL

UTILWS  EQU  >2038
SETWDA  EQU  >24CA
ARGIDS  EQU  >8300
ARGQTY  EQU  >8312
GPLWS   EQU  >83E0
VDPWD   EQU  >8C00

EXAMPL  LWPI MYWS
        MOVB @ARGQTY,R4     # arg's
        SRL  R4,8
        CLR  R5
        LI   R0,->1E
EXA1    CI   R4,0           -¦optional
        JEQ  EXIT           -¦arguments!
        MOVB @ARGIDS(R5),@ARGNOW
        AB   @ASCIIO,@ARGNOW
        AI   R0,>20           next row
        LI   R1,MSG
        LI   R2,15
        BLWP @VMBWBB
        INC  R5
        DEC  R4
        JMP  EXA1

EXIT    LWPI GPLWS
        B    @>006A

VMBWBB  DATA UTILWS,$+2     -¦VMBW
        BL   @SETWDA         ¦with
VWTLOB  MOVB *R1+,R3         ¦BASIC
        AB   @BIAS,R3        ¦bias!
        MOVB R3,@VDPWD       ¦
        DEC  R2              ¦
        JNE  VWTLOB          ¦
        RTWP                -¦

MYWS    BSS  >20

MSG     TEXT 'INDENTIFIER = '
ARGNOW  BYTE 0
BIAS    BYTE >60
ASCIIO  BYTE >30
        EVEN
        END
```

Note the "Video Multiple-Byte Write, BASIC Bias", VMBWBB, routine. It operates like a VMBW, but adds the >6Ø bias for XB. While the VMBWBB is not as efficient in speed or bytes as VMBW with pre-biased text, it does enhance the readability of the source code and is far more efficient than similar routines that utilize VSBW. VSBW resets the VDP address on each call, while the VMBWBB routine takes advantage of VDP's auto-incrementing addressing feature. I haven't tried using VMBWBB with other biases, so you might want to experiment with that!

Here is an XB program that utilizes the above A/L code:

```
> 100 CALL CLEAR !209
> 110 CALL INIT :: CALL LOAD("
  DSK2.ARGID/O")!015
> 120 CALL LINK("EXAMPL",1,"HE
  LLO",A,A$,B(),B$())!044
> 130 CALL KEY(5,K,S):: IF S<1
   THEN 130 ELSE END !217
```

The program will display the argument identifiers for up to the maximum of 16 arguments that can be passed. The number of arguments is optional! Of course, the program is merely an example and serves no real practical purpose.

WHEN AN ARRAY ISN'T!

by Richard M. Mitchell

In the preceding article, I pointed out that argument identifiers could be located in A/L code linked to XB. You may have noticed that as an example of identifiers 4 and 5, I used B() and B$(), respectively. Why not simply use B(3) and B$(3), for instance? Well, TI's protocol for argument identifiers is to consider array elements to be the same as non-arrays. And, yes, some sections of the E/A manual are a bit misleading! Array elements are handled exactly like non-array variables! Here is an A/L routine and the XB XB code to access it to show this point:

```
        DEF  NARRAY

STRASG  EQU  >2010
STRREF  EQU  >2014
GPLWS   EQU  >83E0

NARRAY
        LWPI MYWS
        CLR  R0
        LI   R1,1
        LI   R2,BUFFER
        BLWP @STRREF
```

```
        LI    R2,BUFFE2
        BLWP  @STRASG
        LWPI  GPLWS
        B     @>006A

MYWS    BSS   >20
BUFFER  BYTE  >FF
        BSS   >FF
BUFFE2  BYTE  >03
        TEXT  'BYE'

        EVEN

        END
```

```
> 100 CALL INIT :: CALL LOAD("
  DSK1.NARRAY/O")!118
> 110 A$(3)="HELLO" !204
> 120 CALL LINK("NARRAY",A$(3)
  )!031
> 130 PRINT A$(3)!106
> 140 END !139
```

TRIM & LTRIM, With Arrays Supported!

By Richard M. Mitchell

You've probably guessed by now that the articles on the precedeing pages might be leading up to something. The Assembly program listed below uses some of the techniques described in those articles and adds a few more, including access of multi-dimensional arrays from Assembly!

Extended BASIC has every string function a user could ever need, right? Well, XB is powerful, but there are situations that require a bit more brute force. For instance, strings sometimes begin or end with a character or multiple occurrences of a character that is extraneous. It would be nice to be able to trim those extra characters from the string more quickly than can be done from XB. That's what the Assembly routine listed at the end of this article does! Here's an XB example of accessing the Assembly routines.

```
> 100 CALL INIT !157
> 110 CALL LOAD("DSK1.TRIM/O")
  !195
> 120 DIM A$(2,2)!005
> 130 FOR I=0 TO 2 :: FOR J=1
  TO 2 :: A$(I,J)=RPT$(" ",I+2
  )&"HELLO"&RPT$(" ",I+2):: PR
  INT A$(I,J);LEN(A$(I,J)):: N
  EXT J :: NEXT I !223
> 140 CALL LINK("LTRIM",A$(,),
  " ")!043
> 150 CALL LINK("TRIM",A$(,),"
   ")!222
> 160 FOR I=0 TO 2 :: FOR J=1
  TO 2 :: PRINT A$(I,J);LEN(A$
  (I,J)):: NEXT J :: NEXT I !2
  51
```

When LINKing to TRIM and LTRIM, the first parameter is the string to be trimmed and can be a string variable, single-dimension array or even a multi-dimension array! The second parameter is the character to be trimmed from the string and must always have a length of 0 (a "null string", which has no effect on the trim) or 1 (obviously, 1 is preferred). The program supports either OPTION BASE 1 or OPTION BASE 0 (thanks to a great tip from J. Peter Hoddie -- many, many thanks, Peter!). The program automatically calculates the number of dimensions and the number of elements dimensioned and operates on the entire array extremely quickly. TRIM parses from right to left, truncating the string at the first occurrence of a character other than the specified character (B$ in the example program). LTRIM parses from left to right, eliminating occurrences of a character (again, B$ in the example), until a character other than the specified character is parsed.

TRIM and LTRIM are useful for removing blanks imposed in LINPUTing a FIXED length file, to remove carriage returns and line feeds from the ends of a series of strings, to remove the extraneous "0"'s that are sometimes derived at the end of a string while using CHARPAT, to remove characters resulting from conversions between strings and numbers, etc. It might be interesting to see what sorts of games, graphics, etc. might be possible using these routines. The Assembly routines occupy only 594 bytes!

The April 1984 issue of *The Smart Programmer*, page 10, describes the make-up of the Symbol Table, describing in detail the byte structure that the following routines access. You may also want to refer to the Extended BASIC Scratchpad Map in the August, 1986 issue. And, refer to the material in the BASIC Support section of the E/A manual for information on array access. I hope everyone enjoys this article because if it weren't for this, this issue likely

would have been completed a long time ago! Many thanks go to D.C. (Doug) Warren for generously sharing his knowledge with me and for writing Explorer, which, along with a lot of patience, made this article possible.

```
       DEF  TRIM,LTRIM
* PARM1=STRING,PARM2=CHAR,PARM3=BASE

STRASG EQU  >2010
STREF  EQU  >2014
VMBR   EQU  >202C
ARGID1 EQU  >8300
ARGS   EQU  >8312
BASE   EQU  >8343 Thanks, P. Hoddie!
VSTKPT EQU  >836E
GPLWS  EQU  >83E0

TRIM   MOVB @ONE,@FLAG
LTRIM  LWPI MYWS
       CLR  R0          |
       LI   R1,2        | CHARACTER
       LI   R2,LEN2     | TO TRIM
       BLWP @STRREF     |
       LI   R8,1
       CB   @LEN2,@ZERO | NULL?
       JNE  TR1         |
       B    @EXIT2
TR1    CLR  R8
       CB   @ARGID1,@FIVE ARRAY?
       JNE  TR4
       MOVB @ARGS,R3    | OFFSET TO
       SRL  R3,8        | 1ST STACK
       DEC  R3          | POINTER
       SLA  R3,3        | ENTRY
       MOV  @VSTKPT,R0    STK PTR
       S    R3,R0         OFFSET
       LI   R1,STACKA   | STK ENTRY
       LI   R2,2        | ADDRESS
       BLWP @VMBR
       MOV  @STACKA,R0
       LI   R1,DIMS       DIMS+80
       LI   R2,1
       BLWP @VMBR
       SB   @OFFSTD,@DIMS OFFSET
       MOVB @DIMS,R4    |
       SRL  R4,8        |
       AI   R0,4        | FIGURE
       LI   R1,REG9     | TOTAL
       LI   R2,2        | # OF
       LI   R7,1        | ELEMENTS
LOOP   INCT R0          |
       BLWP @VMBR       |
       CB   @BASE,@ZERO |
       JNE  TR2         |
       INC  R9          |
TR2    MPY  R9,R7       |
       MOV  R8,R7       |
       DEC  R4          |
       JNE  LOOP        |
TR3    MOV  R8,R0
       CB   @BASE,@ZERO   BASE 0?
       JNE  TR4
       DEC  R0            ZERO OK
TR4    LI   R1,1        | STRING
       LI   R2,LEN1     | TO
       BLWP @STRREF     | TRIM
       CLR  R6            |
       MOVB @LEN1,R3      |
       SRL  R3,8          | PARSE
       CB   @ONE,@FLAG    | STRING
       JEQ  TRIM1         | AND
       LI   R2,1          | MARK
       JMP  T1            | FOR
TRIM1  MOV  R3,R2         | TRIM
T1     CB   @LEN1(R2),@CHR2 |
       JNE  EXIT          |
       CI   R3,0          |
       JEQ  EXIT          |
       DEC  R3            |
       CB   @ONE,@FLAG    |
       JEQ  T2            |
       INC  R2            |
       INC  R6            |
       JMP  T1            |
T2     DEC  R2            |
       JMP  T1            |

EXIT   LI   R1,1                \
       MOVB @LSB3,@LEN1(R6)     | WRITE
       LI   R2,LEN1             | TRIMMED
       A    R6,R2               | STRING
       BLWP @STRASG           /
       MOVB @MAX1,@LEN1  | PREP FOR
EXIT2  MOVB @ONE,@LEN2   | NEXT
       CB   @ARGID1,@FIVE ARRAY?
       JNE  RETURN
       DEC  R8
       JNE  TR3
RETURN MOV  @ONE,@LEN3   | PREP FOR
       MOVB @ZERO,@FLAG  | NEXT LINK
       LWPI GPLWS
       B    @>006A

MYWS   BSS  >20  WORKSPACE
LSB3   EQU  MYWS+7   LSB OF R3
REG9   EQU  MYWS+>12 MSB OF R9
STACKA DATA 0    VALUE STACK ADDR
LEN1   BYTE >FF  LEN OF STRING
       BSS  >FF  STRING TO TRIM
LEN2   BYTE 1    LEN OF CHARACTER
CHR2   BYTE 0    CHAR TO TRIM
LEN3   BYTE 1    LEN OF BASE
CHR3   BYTE 0    OPTION BASE
OFFSTD BYTE >80  DIM OFFSET
MAX1   BYTE >FF  MAX LENGTH
FLAG   BYTE 0    0=LTRIM,1=TRIM
ZERO   BYTE 0    | #'s FOR BYTE
ONE    BYTE 1    | COMPARISONS
```

```
FIVE    BYTE 5      |
DIMS    BYTE 0      # OF DIM'S
        EVEN

        END
```

In retrospect, it looks rather simple. I guess that's the difference between hindsight and foresight! It's really interesting that as a program improves, it often gets smaller!

Quote

We should market to our friends, not people who don't like our style of computing.

Jean-Louis Gassée, Apple V.P., in *Lotus* magazine.

Write GRAM!

program by Mike Dodd
article by Richard Mitchell

Here's an extremely useful XB Assembly routine for Gram Kracker™ owners. The advantages of this program are that it allows you to write to the write-protected GRAM's, 3-7, and allows writing an entire string at a time! See the article that follows this one for an example of the XB usage of the program. Note that byte values above 32767 must be converted to a negative number by subtracting 65536, as with GK Util I's PEEKG and POKEG. Many thanks to Mike Dodd for this outstanding program!

```
        DEF  WRTGRM
GWA     EQU  >9C02
GRA     EQU  >9802
GWD     EQU  >9C00
NUMREF  EQU  >200C
STRREF  EQU  >2014
FAC     EQU  >834A
HFF     BYTE >FF
BANK1   TEXT 'Enable bank 1&press FCTN'
BANK0   TEXT 'Restore W/P & press FCTN'
        EVEN
PBASIC  DATA SUBWS1,PBAS1
PBAS1   MOVB *R13,R0
        MOVB @1(R13),@>8C02
        ORI  R0,>4000
        MOVB R0,@>8C02
        MOV  @2(R13),R0
        MOV  @4(R13),R1
PBAS2   MOVB *R0+,R2
        AI   R2,>6000
        MOVB R2,@>8C00
        DEC  R1
        JNE  PBAS2
        RTWP
WRTGRM  LWPI MYWS
        MOVB @>8312,R6
        JEQ  RETURN
        SRL  R6,9
        LI   R0,>184
        LI   R1,BANK1
        LI   R2,24
        BLWP @PBASIC
        CLR  R12
FCTN1   TB   7
        JEQ  FCTN1
        CLR  R8
A       CLR  R0
        INC  R8
        MOV  R8,R1
        BLWP @NUMREF
        LWPI >83E0
        BL   @>12B8
        LWPI MYWS
        MOV  @FAC,R9
        CLR  R0
        INC  R8
        MOV  R8,R1
        LI   R2,BYTESL
        MOVB @HFF,*R2
        BLWP @STRREF
        MOVB @GRA,R7
        SWPB R7
        MOVB @GRA,R7
        SWPB R7
        DEC  R7
        MOVB R9,@GWA
        SWPB R9
        MOVB R9,@GWA
        MOVB @BYTESL,R9
        SRL  R9,8
        LI   R0,BYTES
B       MOVB *R0+,@GWD
        DEC  R9
        JNE  B
        MOVB R7,@GWA
        SWPB R7
        MOVB R7,@GWA
        DEC  R6
        JNE  A
FCTN2   TB   7
        JNE  FCTN2
        LI   R0,>184
        LI   R1,BANK0
        LI   R2,24
        BLWP @PBASIC
FCTN3   TB   7
        JEQ  FCTN3
RETURN  LWPI >83E0
        B    @>6A
```

```
SUBWS1  DATA 0,0,0,0,0,0,0,0
        DATA 0,0,0,0,0,0,0,0
MYWS    DATA 0,0,0,0,0,0,0,0
        DATA 0,0,0,0,0,0,0,0
BYTESL  BYTE 0
BYTES   BSS  255
        END
```

Seven New XB CALL's

Code by Mike Dodd
Article by Mike Dodd and Richard M. Mitchell
Implementation by Richard M. Mitchell

Mike Dodd has developed seven new CALL's for users of MG's GK Utility I version of Extended BASIC!

Because seven CALL's represents more data than we typically cover, we'll take a different approach to implementing the CALL's. Rather than key the data directly, risking an irrecoverable error, we'll use a program to checksum the data and write it to GRAM. And, you'll have a choice as to whether you want to use Mike Dodd's WRTGRM program. If you choose not to use WRTGRM, you'll use POKEG, which cannot be safely used to write to GRAM 6, where the code will end up, so we'll write to GRAM 2 and then move it.

The CALL's are as follows:

CALL BEEP -- produces a beep tone.
CALL HONK -- produces a honk tone.
CALL STSPRT -- stops all sprite motion. Note that sprite motion remains disabled even after the program is run (while the console is powered up), so follow CALL STSPRT with CALL GOSPRT before the end of your program.
CALL GOSPRT -- reverses CALL STSPRT, enabling sprite motion.
CALL SCROFF -- disables all screen displays (the same thing as happens when the screen times out when no key has been depressed).
CALL SCRON -- enables the screen, reversing CALL SCROFF.
CALL COLORS(F,B) -- Sets color sets Ø through 14 to foreground color F and background color B. This is similar to the XB routine FOR X=Ø TO 14 :: CALL COLOR(X,F,B) :: NEXT X. This CALL does it much faster. For maximum flexibility, the border color of the screen is not affected. The border color can be changed using CALL SCREEN(Z). If B in CALL COLORS is set to Ø (transparent), the background color will appear the same as the color specified in the CALL SCREEN command.

Note: If you have added your own code, be sure you have not used >D8FB through >D9C4 in GRAM, as that is where this modification will reside.

To install the changes, be sure to follow these instructions very carefully:

1) Be sure the contents of your GRAM 2 are saved to disk (for the POKEG installation method, the area that GK Utility I leaves free beginning at >52Ø8 will be used for temporary storage of 2Ø7 bytes).

2) A) Key in the following program and save it to disk if you will not be using Mike Dodd's WRTGRM program (see 2B for the modifications for the WRTGRM program):

```
> 100 DIM A(208)!157
> 110 FOR I=1 TO 208 :: READ A
  $ :: CALL HEX_DEC(A$,D):: A(
  I)=D :: N=N+A(I):: NEXT I !0
  88
> 120 IF N<>40018 THEN PRINT "
  DATA INTEGRITY ERROR" :: END
   !242
> 130 A(1)=A(1)-1 :: FOR I=1 T
  O 207 :: CALL POKEG(A(1)+I,A
  (I+1)):: NEXT I !053
> 140 END !139
> 1000 DATA 5208 !046
> 2000 DATA 06,D8,FB !193
> 3000 DATA D9,14 !188
> 4000 DATA 86,A3,70,86,8F,FC,
  FA,BD,00,8F,ED,00 !078
> 4010 DATA 86,8F,FC,FC,D5,00,
  8F,ED,00,59,13,0B !062
> 4020 DATA 00 !189
> 5000 DATA D9,1D,04,42,45,45,
  50,D9,5C,D9,26 !012
> 5010 DATA 04,48,4F,4E,4B,D9,
  62,D9,31,06,53,54 !242
> 5020 DATA 53,50,52,54,D9,68,
  D9,3C,06,47,4F,53 !234
> 5030 DATA 50,52,54,D9,6F,D9,
  47,06,53,43,52,4F !232
> 5040 DATA 46,46,D9,78,D9,51,
  05,53,43,52,4F,4E !237
> 5050 DATA D9,85,00,00,06,43,
  4F,4C,4F,52,53,D9 !234
> 5060 DATA 9A,06,00,34,06,00,
  12,06,00,36,06,00 !129
> 5070 DATA 12,B6,80,C2,40,06,
  00,12,B2,80,C2,BF !218
> 5080 DATA 06,00,12,A0,E0,39,
  00,01,01,D9,76,BE !202
> 5090 DATA 80,D4,A0,06,00,12,
  39,00,01,01,D9,77 !175
> 5100 DATA BE,80,D4,E0,06,00,
  12,0F,79,0F,74,B6 !000
```

EXPLORER

YOUR WINDOW INTO THE 99/4A

Gaze into the inner workings of your 99/4A with EXPLORER. Just load it and with a single keystroke EXPLORER will start up where your console, application program or module left off, but YOU will be in FULL CONTROL! Watch EXPLORER's screens, with dynamic information, or flip to the ACTUAL Program Screen running in slower motion under YOUR CONTROL. Track, Display, Edit and Search VDP Memory, CPU Memory or GROM/GRAM Memory. Set breakpoints for pausing execution at ROM, RAM, VDP, GROM or GRAM addresses. EXPLORER displays the current Registers, GPL Status and VDP Registers. And, each machine instruction is disassembled. EXPLORER's Options Screen provides arithmetic and logical operations in Decimal, Hex and Binary.

Sounds exciting, right? Well, the best part is that EXPLORER is now better than ever. EXPLORER now loads through standard module loaders, allowing you to load the program from RAMdisk, floppy disk or hard disk. And, a special transparent loader is provided for quick loading from XB.

System requirements: 99/4A with Disk System, Memory Expansion and a standard loading environment such as Extended BASIC, Editor/Assembler, Mini Memory or Gram Kracker™

Only $24.95

™ Gram Kracker is a Trademark of MG

WORDS WERE JUST DATA UNTIL...

STRING MASTER

A POWERFUL STRING MANIPULATION PACKAGE FOR THE 99/4A

STRING MASTER provides the string functions Extended BASIC users have always wanted. Most of STRING MASTER's functions work on string variables or an entire string array. TRIM or LTRIM unwanted leading or trailing characters. CONCTL or CONCTR concatenates any combination of string variables and string arrays. That's right, concatenate the elements of one array to the elements of another array in a single Extended BASIC statement! POKE and PEEK string variables to and from VDP or CPU memory. SEARCH a string variable or string array for a word or phrase. Many other features!

Available June, 1987

System Requirements: 99/4A, Disk System, Memory Expansion, Extended BASIC

Only $19.95

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 7Ø663

Please add $2.ØØ shipping and handling per software order
Payments accepted by personal check or money order.
Foreign payments by international money order, please.

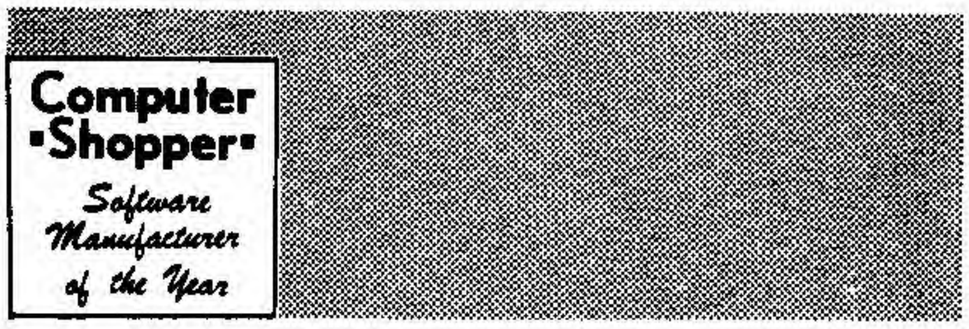
Asgard Software

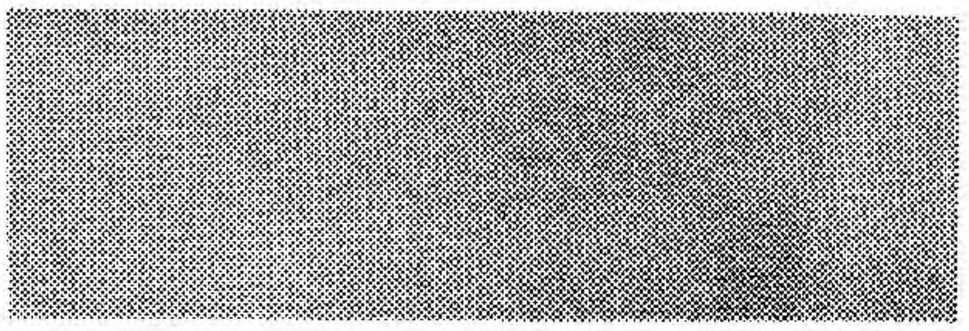
is proud to present a piece of the future:

The first two commercial programs written in c99 for the 99/4A — The fastest language for the 99/4A outside of assembly!

High Gravity

- Is **High Gravity** an educational game or a game program that's educational? Who knows which, and it really doesn't matter considering that this incredible simulation written in c99 (a language faster than Forth and easier to use than BASIC) is one of the best programs ever written for the 99/4A in any language!
- **High Gravity**, by Tom Wible (a professional programmer), puts you in command of a relief spacecraft sent to aid a space station trapped in a strange solar system. The planets in the system are thick as flies, and prevent anyone from leaving or entering the solar system to rescue the unfortunate people in the space station. Your mission is to shoot a capsule of supplies to the stranded astronauts, and you only have ten capsules of supplies on hand. Worse yet, you can't guide the capsules through since they have no engines. Fantastic graphics make this game colorful as well as exciting.
- **High Gravity** is also an extremely accurate simulation of the Laws of Gravity and the motion of projectiles. The fact that this program is a sophisticated lesson on physics is not apparant — it's a really fun game that gives hours of enjoyment to children AND adults. However, for the educational user all varibles of the program may be pre-set; including the initial velocity, the density, size, and spacing of the planets, and much more. High Gravity will even let you save and load interesting flight paths of projectiles for later study — a library of such paths is included with the program.
- In short, **High Gravity** is a sophisticated simulation of space flight that is both entertaining and educational. It is an ideal teacher for the physics student (of all levels), and an ideal game for all ages.





It is simple to use and fully documented. It requires the Editor/Assembler module, 32 K and a disk system. Available for only $14.95.

Total Filer

- Do you have disks and disks full of TI-Writer text files cluttering up your disk library? Do you often catalog one of your TI-Writer disks and find files that you didn't know you had, or even know what they are? Well then, we would like to introduce to you the greatest tool for user's of TI-Writer since the spelling checker; the first and only database designed for text — **Total Filer** by Warren Agee.
- Some database programs say they will let you organize anything, but nothing matches the speed, power and flexibility of a program exclusively designed to let you organize text when it comes to organizing your TI-Writer files. **Total Filer** is a very easy-to-use solution for a complex problem. It is written in c99, an incredibly fast language for the 99/4A, and was designed specifically for handling text.
- With **Total Filer** you can easily create a file-by-file reference of all your text files. Your index can include multiple keyword references for quick searches, as well as several layers of keywords for in-depth descriptions. For searching, **Total Filer** even includes utilities for creating a master listing of the index, as well as letting you compress it to save space on your data disks. **Total Filer** is truly a tool for the "power user."
- **Total Filer** is also very flexible, allowing users to do everything from configure the program for any hardware combination to setting the names of the prompts for different functions. **Total Filer** is the penultimate tool for organizing text of any sort, from magazine articles to computer files, yet it is easy to use and fully documented. It requires the Editor/Assembler module, 32K and a disk system. Available for only $24.95.

Asgard Software

P.O. Box 10306
Rockville, MD 20850
(301) 345-2492

"Serving the TI Community"

Note: c99 compiler for the 99/4A by Clint Pulley

```
> 5110 DATA 0F,12,00,06,A9,D6,
  06,D9,92,93,4A,BD !004
> 5120 DATA 00,4A,E3,00,00,0C,
  06,D9,92,93,4A,E3 !227
> 5130 DATA 4A,00,08,B4,00,4A,
  BC,A8,00,00,35,00 !202
> 5140 DATA 1F,A8,01,A8,00,0F,
  79,06,00,12 !251
> 30070 SUB HEX_DEC(H$,D):: D=
  0 :: L=LEN(H$):: FOR I=1 TO
  L :: P=POS("0123456789ABCDEF
  ",SEG$(H$,I,1),1)-1 :: D=D+P
  *16^(L-I):: NEXT I :: SUBEND
   !185
```

B) If you want to use WRTGRM (and not have to manually move bytes from the GK Editor), omit lines 100 through 1000 above and add these lines to the above program:

```
> 100 CALL INIT :: CALL LOAD("
  DSK1.WRTGRM/O"):: DISPLAY AT
  (7,1)ERA^_ ALL:"NEXT PROMPT
  IN ABOUT         70 SECONDS"
  !095
> 110 READ A$,A,B$,B,C$,C,D$,D
   :: CALL HEX_DEC_M(A$,A1)::
  CALL HEX_DEC_M(B$,B1):: CALL
   HEX_DEC_M(C$,C1):: CALL HEX
  _DEC_M(D$,D1)!184
> 120 CALL PREPWRT(A,W1,W1$)!2
  32
> 130 CALL PREPWRT(B,W2,W2$)!2
  35
> 140 CALL PREPWRT(C,W3,W3$)!2
  38
> 150 CALL PREPWRT(D,W4,W4$)!2
  41
> 160 IF (W1+W2+W3+W4)<>19018
  THEN PRINT "DATA INTEGRITY E
  RROR" :: END !128
> 170 DISPLAY AT(15,1)ERASE AL
  L:"DO YOU HAVE A MYARC EXTEN
  DEDBASIC PROM INSTALLED?":"Y
  " !091
> 180 ACCEPT AT(17,1)BEEP VALI
  DATE("YN")SIZE(-1):R$ !232
> 190 IF R$="Y" THEN CALL LINK
  ("WRTGRM",B1,W2$,D1,W4$)ELSE
   CALL LINK("WRTGRM",A1,W1$,B
  1,W2$,C1,W3$,D1,W4$)!158
> 200 END !139
> 1000 DATA 6372,3,D789,2,D8FB
  ,25,D914,177 !222
> 30080 SUB HEX_DEC_M(H$,D)::
  D=0 :: L=LEN(H$):: FOR I=1 T
  O L :: P=POS("0123456789ABCD
  EF",SEG$(H$,I,1),1)-1 :: D=D
  +P*16^(L-I):: NEXT I !061
> 30090 IF D>32767 THEN D=D-65
  536 !236
> 30100 SUBEND !168
> 30110 SUB PREPWRT(A,W1,W1$)!
  236
> 30120 FOR I=1 TO A :: READ Z
  $ :: CALL HEX_DEC(Z$,Z):: W1
  =W1+Z :: W1$=W1$&CHR$(Z):: N
  EXT I :: SUBEND !239
```

3) Run the program. If "DATA INTEGRITY ERROR" is printed on your screen, then the checksum total is in error, indicating you have keyed the DATA and/or program in improperly. If you used the WRTGRM method in 2B above, do not perform the actions in steps 4 through 9, though you may want to read step 5.

4) Switch to the Gram Kracker™ editor. Switch write protect off. Press <FCTN 1> until you are in the G(RAM) window.

5) This is where things get a bit sticky. If you have a MYARC XB PROM installed, skip to Step 8. Steps 6 and 7 cause a return to the Power-up Title Screen if the W/P switch on the GK is disabled. However, the MYARC PROM has power-up priority and may load data from the PROM if the W/P switch is disabled (when this happens, you'll see "128K O.S." on your Main Menu). Checking for the position of the W/P switch through Mike's routine is very useful, as XB will go "out to lunch" if W/P is disabled because XB has 2 banks of ROM that are banked by doing a pseudo-write to ROM, so that if an actual write is done, XB will not bank and will be left in the wrong bank of ROM for the current activity. MYARC XB owners will have to continue to visually inspect the W/P switch. Note that WRTGRM is not affected by the MYARC PROM because there is no power-up during the execution of WRTGRM.

6) Set the following values for a MOVE:

```
 START FINISH DEST
 5208  520A   g6372
```

Press <FCTN 2>.

7) Set the following values for a MOVE:

```
 START FINISH DEST
 520D  5225   gD8FB
```

Press <FCTN 2>.

8) Set the following values for a MOVE:

```
 START FINISH DEST
 520B  520C   gD789
```

Press <FCTN 2>.

Note: If you have added your own CALL's, change DEST gD789 to the address of the end of your link table for subprogram CALL's, which can be determined by using Subprogram Finder.

9) Set the following values for a MOVE:

```
START FINISH DEST
5226  52D6   gD914
```

Press <FCTN 2>.

1Ø) If you have the MYARC XB PROM, be absolutely certain your W/P switch is on W/P!

11) Save the revised module to disk, using a filename different from the previous XB filename.

12) Run Mike Dodd's Subprogram Finder program if you wish.

13) Test the new CALL's.

TECHNICAL INFORMATION:

CALL BEEP and CALL HONK operate by using GPL routines >34 and >36, respectively.

CALL STSPRT and CALL GOSPRT operate by setting and resetting, respectively, bit 1 at >83C2.

CALL SCROFF and CALL SCRON operate by setting and resetting, respectively, bit 1 of VDP Register 1.

Important GRAM addresses:

GRAM 6

>D951 (>ØØ,>ØØ). This is the new end of the link table used by subprogram CALL's in XB.

There are now 1,571 free bytes of memory in GRAM 6 from >D9C5 through >DFE7.

Oops!

Well, due to a goof by the Editor, last issue's GK Menu article didn't cover the new revisions for the alpha menu. Here are Tom Freeman's revisions:

1) Search for BE 58 3Ø and change the 3Ø to 4Ø.
Your search should locate at about gØ275.

2) Search for A6 75 31 and change the 31 to 41.
Your search should locate at about gØ2FC.

3) Save the revised GRAM Ø.

An additional note is that if you are using Gram Packer to create menu's, you should plan the number of menu items to be 16 minus the maximum number of menu options of a loaded module. For instance, if the Navarone Database Manager is to be loaded, it will generate 4 menu options, so there should be no more than 12 items on the original main menu.

The 8ØSYLK program in *Super 99 Monthly* (and on the Best of *Super 99 Monthly* diskettes) used an early version of the R_A_W assembly program, so it may not work with some (not all) MYARC disk controllers. XXB/1-2, as appeared in the *Genial TRAVelER* diskazine, includes an improved R_A_W routine and should alleviate any problems experienced with the 8ØSYLK program. The R_A_W modification involved changing the location of a buffer in VDP.

BasicSort Version 2

A special report by Richard M. Mitchell

BasicSort is an excellent program. The program will sort numerics, strings or string segments and will perform up to 16 levels of sorts with a single program statement! BasicSort is written in Assembly, so it is extremely fast and is ideal for use with your BASIC or Extended BASIC programs. The documentation is thorough and well-written, on an intermediate level. A better bargain is not available in computerdom, as BasicSort is only $15 plus $3 shipping and handling from Andreas L. Dessoff, 1Ø41 Church Hill Road, Fairfield, CT Ø6432.

E/A Enhancements

Code by Craig Miller
Article by Richard M. Mitchell

Craig Miller has used DISkASSEMBLER™ to disassemble the EDIT1 file of the Editor/ Assembler and has come up with some useful modifications. The instructions that follow are for implementation for GK Utility I E/A version, but may be modifiable for other implementations of E/A.

1) Using the GK Memory Editor, search for:

Ø6 FF ØØ Ø3

I found the above at g7AA2. Change it to:

Ø3 FF ØØ Ø3

This is part of the delay before a key goes into auto-repeat. You can also change the Ø3 to any number from Ø1 through ØF to make the cursor blink faster (Ø1) or slower (ØF) (I like Ø5).

2) Search for:

ØA ØØ Ø6 ØØ

I found this at g7BB4. Change it to:

ØØ Ø1 Ø6 ØØ

This is the delay loop between keystrokes.

With the above changes in place you will notice that the cursor moves quite a bit faster and goes into auto-repeat quite a bit faster.

After making the changes, you'll need to follow the instructions on page 22 of the GK Utility I manual to change the checksum for your new E/A.

Some other interesting addresses (based on the files I'm using) for possible changes are:

g981C-g9826	End of File marker.
g9E10	Command Line text.

You may also want to search for the default Tabs (they're offset by minus one).

On-line Conference

On May 16, Los Angeles (USA), Ottawa (Canada) and Derby (England) will hold 99/4A Fairs. The 3 shows will be linked via an on-line conference on GEnie™. The event is set for 1Ø AM in LA, 1 PM in Ottawa and 6 PM in Derby. Figure out what time it will be in your area and you can join in!

Triton XB

Triton Products Company has announced Super Extended BASIC, which adds many new features to XB. Included are the XB enhancements that appeared in MG's GK Utility I package (see *The Smart Programmer*, July 1986 for features), the code of Mike Dodd that appears in this issue, plus CALL ALL(x), fills screen with a character; CALL CHIMES, chimes sound; CALL GOSUB(num var), allows numeric variable; CALL GOTO(num var), allows numeric variable; CALL KEYS("keylist",num var), allows valid key list; CALL ALOCK(x), checks alpha lock key; CALL SHIFT(x), checks shift key; CALL CTRL(x), checks control key; CALL FCTN(x), checks function key. Two other enhancements were pending at press time. CALL VERSION will return 12Ø instead of the previous 11Ø (or the 1ØØ of the original XB). The package is 1ØØ% compatible with all TI XB programs. The new package is priced at $59.95 in the Spring Triton catalog.

In other news from Triton, the firm has now shipped its Triton Turbo XT computer (see *The Smart Programmer*, November, 1986), a PC clone that can interface with the 99/4A keyboard or a standard XT keyboard.

"Inverted Mouse": Reprise

by Richard M. Mitchell

A number of you have asked how to add a keyboard scan (for the keys "1" through "4") to the "Mouse" XB program that appeared in the January 1985 issue of *Super 99 Monthly* (and the *S99M* disks). Simply relocate the sprite to a position next to the appropriate number, as follows:

```
21040 CALL KEY(2,K1,S):: CAL
L KEY(1,K,S):: IF K1=18 OR K
=18 THEN 21046 !150
21044 IF K=19 THEN K=K-13 EL
SE IF K<7 OR K>9 THEN 21020
!114
21045 X=(K-5)*16+9 :: CALL L
OCATE(#1,X,40)!058
21046 CALL POSITION(#1,X,Y)!
093
```

For those of you who haven't caught on, an "inverted mouse" is a joystick! Ha!

BYTEMASTER ORDER FORM

The Smart Programmer

SP1	$18.00	U.S. AND CANADA FIRST CLASS
SP2	$15.00	U.S. THIRD CLASS (no back issues)
SP3	$20.00	FOREIGN SURFACE (no back issues)
SP4	$32.00	FOREIGN AIRMAIL
SP5A-G	$ 1.75	U.S. JUNE - DEC 1986, ea.
SP6A-G	$ 2.75	FOREIGN JUNE - DEC 1986, ea.
SP7A-G	$ 2.50	U.S. JAN 84-AUG 84,photostats,ea.
SP8A-G	$ 3.50	FOREIGN JAN 84 - AUG 84, ea.

Super 99 Monthly

SM1	$18.00	Complete set of 18 back issues
SM2A-R	$ 1.00	Back issues - ea. (U.S. Third Class)
SM3A-R	$ 1.50	Back issues - ea. (Canada and U.S. First Class)
SM6A-R	$ 2.50	Back issues - ea. (Foreign Air Mail)
SM4	$12.00	Programs on disk (non-FORTH)
SM5	$15.00	Super 99 Handicapper (thoroughbreds) (req. XB, 32K, Disk, Printer)

ITEM #	QTY	EACH	AMOUNT
_____	_____	_____	_____
_____	_____	_____	_____

New +-+ | | +-+ Renewal +-+ | | +-+

NAME _____

ADDRESS _____

CITY _____

STATE _____

ZIP CODE _____

COUNTRY _____

Payments accepted by check or money order in U.S. funds, coded for processing through the U.S. Federal Reserve Banking System. No billings or credit sales. Dealer inquiries invited. Discounts available on quantity orders.

Louisiana residents must add 4% sales tax. Calcasieu 1%. Sulphur 2%.

The Smart Programmer is published monthly by Bytemaster Computer Services, 171 Mustang Street, Sulphur, LA 70663.

Editor Richard M. Mitchell

Staff Craig Miller, Steven J. Szymkiewicz, MD, Charles M. Robertson, Barry A. Traver, Mariusz Stanczak, D.C. Warren

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663-6724
U.S.A.

FIRST CLASS MAIL

POSTMASTER: ADDRESS CORRECTION REQUESTED
RUSH -- TIME DATED MATERIAL

FIRST CLASS MAIL